IP Project Gearup II
Overview

• How to think about forwarding/link-layer
• How to debug/view in wireshark
• Implementation notes
• Any questions you have
The Big Picture

User command line (REPL)

User commands

Other commands
(up, down, l*, …)

TCP
(next project, hosts only)

Routing
(RIP)
(routers only)

Test packets

Upper layers

IP API

IP Forwarding

“Network layer”

if0
UDP Socket

if1
UDP Socket

“Link layer”
(Interfaces)

User commands
What you should be focusing on first

User commands

User command line (REPL)

Upper layers

Test packets
Routing (RIP) (routers only)
TCP (next project, hosts only)

"Network layer"

IP API
IP Forwarding

"Link layer" (Interfaces)

if0
UDP Socket

if1
UDP Socket

...
What you should be focusing on first

How to receive packets on interfaces, send them back out
How does the link-layer work?

What does it mean to forward vs. send on an interface?

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next hop</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/24</td>
<td>LOCAL:if0</td>
<td>0</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>10.0.0.2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next hop</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.0.0/24</td>
<td>10.1.0.2</td>
<td>1</td>
</tr>
<tr>
<td>10.0.0.0/24</td>
<td>LOCAL:if0</td>
<td>0</td>
</tr>
<tr>
<td>10.1.0.0/24</td>
<td>LOCAL:if1</td>
<td>0</td>
</tr>
</tbody>
</table>
Node ::= “host” or “router”

All nodes connect via interfaces
⇒ Hosts have exactly one interface
⇒ Routers have multiple interfaces

<table>
<thead>
<tr>
<th>T</th>
<th>Prefix</th>
<th>Next hop</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>10.2.0.0/24</td>
<td>10.1.0.2</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>10.0.0.0/24</td>
<td>LOCAL:if0</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>10.1.0.0/24</td>
<td>LOCAL:if1</td>
<td>0</td>
</tr>
</tbody>
</table>
All topologies have multiple subnets:
- Each subnet has its own IP prefix
- Each interface is connected to one subnet
- Nodes on the same subnet are neighbors
  => Nodes always know how to send packets to their neighbors
h1.lnx

```
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```

Virtual IP: 10.0.0.1
Network: 10.0.0.0/24
UDP: bind on 127.0.0.1:5000
```
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```

**Config for if0**

- **Virtual IP:** 10.0.0.1
- **Network:** 10.0.0.0/24
- **UDP:** bind on 127.0.0.1:5000
- **neighbors:** {10.0.0.2 => 127.0.0.1:5001}

*ONE FOR EACH OTHER NODE ON THIS SUBNET*

*EACH INTERFACE HAS A SET OF NEIGHBORS CAN ALWAYS SEND DIRECTLY TO NEIGHBORS.*
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2

Virtual IP: 10.0.0.1
Network: 10.0.0.0/24
UDP: bind on 127.0.0.1:5000
neighbors: { 10.0.0.2 => 127.0.0.1:5001 }

Each interface has a list of neighbors: mapping of IPs to UDP ports
=> Like an ARP table, but always known ahead of time
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2

Virtual IP:  10.0.0.1
Network:  10.0.0.0/24
UDP:  bind on 127.0.0.1:5000
neighbors:  { 10.0.0.2 => 127.0.0.1:5001 }

=> H1 can reach 10.0.0.2 by sending to UDP port 5001

So if we want to send from H1 to R1, we need to send something to UDP port 5001 => but what?
How to think about encapsulation

- Each interface: thread/goroutine/etc listening on a UDP port
- Each packet contains an IP header + whatever message content

**What is sent on UDP socket**

```
[IP HEADER] --- "HELLO"
```
IP Header

How we "unwrap" the packet.
UDP-in-IP example

- Complete code example for building an IP header, adding it to a packet, and sending it via UDP
  - Also computes/validates checksum!

- Let’s break down how this works…
To send some data

• Build an IP header
  – Fill in all header fields as appropriate (source, dest IP, etc.)
  – Compute the checksum
• UDP Packet: IP header + data you want to send
• Send packet via socket for that interface
What would it look like to send from h1 -> h3?

```
10.2.0.2 => LOCAL => :5005
10.0.0.1 => NA 10.2.0.1 => :5004
```
What happens if h2 sends to h3?

If sending from h3...

<table>
<thead>
<tr>
<th>Dest</th>
<th>Fwd Table</th>
<th>Neighbors Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>10.2.0.2</td>
<td>10.2.0.2 ➔ :5005</td>
</tr>
<tr>
<td>10.2.0.2</td>
<td>10.0.0.1</td>
<td>10.2.0.1 ➔ :5004</td>
</tr>
</tbody>
</table>

N2 is neighbor, so can send directly to host!
Receiving packets

• Receive packet from link layer
• Parse IP header and determine if packet is valid
  – TTL, checksum, etc…
• Check destination IP
  – If destination is your IP: deliver locally
  – If not, consult forwarding table
FORWARDING STEPS:

**Consider packet with destination IP D**

If destination IP D matches one of this node’s assigned IPs
  => Packet is for this node  => Send “up” (more on this later)

Otherwise, check forwarding table to look for a match
(If multiple matches, take the most specific prefix (lecture 7, 9)

If the result is a local route (ie, maps to some ifX)
  => Look up UDP port for D in neighbors table for ifX
  Send packet to this port

If the result is not a local route (ie, has next hop IP G)
  => Need to send packet to G instead:
    Look up G in forwarding table
    => maps to some local route on some interface ifY
    Look up UDP port for G in ifY’s neighbor’s table
    Send packet to this port

Choosing NEXT-hop DESTINATION!
How to send "up"?

Our nodes do different things with packets:

- Test packets (0)
- TCP (6)
- Routes
- RIP packets (2)
- TCP packets (6)

Look up a handler for this packet based on protocol num:

\texttt{REGISTER HANDLER (num, somefunc)}

Do this at startup — tell IP stack to call somefunc when receiving a packet w/ this protocol.
How to table lookup?

Dest IP == 10.0.0.5, where to send packet?

- You can decide how to store the table
- Need to find the most specific matching prefix
- Use built-in datatypes to help you!
  Go: `prefix.Contains()` (netip.Prefix)

You do NOT need to be particularly efficient about this step!
Implementation: key resources

• Use an external library for parsing IP header (don’t do this yourself)
  – For Go/C, see UDP-in-IP examples
  – Rust: etherparse library

• We provide parsers for the Inx files—don’t bother writing these yourself

• You’re welcome to use third-party libraries, so long as they don’t trivialize the assignment (ask if you’re concerned)
  – Data structures, argument parsing, are fine
IP types and go

Go has two IP types, net.IP and (newer) netip.Addr

- netip.Addr and netip.Prefix the one you want

⇒ These libraries have useful helper functions, use them!
vnet_run: Run all nodes in a network automatically
• Can run on your node, or the reference
• Uses tmux: see getting started guide for details
• Can run some nodes as reference, some nodes as yours

⇒ See getting started guide for details, more soon!
Viewing packets in wireshark
Sample Topologies

Some example networks you can test with...

See “sample networks” page for more info, including what kinds of things you can test with each network.
Roadmap

Once you can send across one router, start thinking about RIP

3. Make sure you can share routes and update the forwarding table
   - Eg. linear-r2h2: H1 -> R1 -> R2 -> H2

4. Try disabling/enabling links, make routes expire

5. Loop network: finding best path, updating routes as topology changes